# Efficient Symmetry Breaking for Boolean Satisfiability 

Fadi A. Aloul, Member, IEEE, Karem A. Sakallah, Fellow, IEEE, and Igor L. Markov, Senior Member, IEEE


#### Abstract

Identifying and breaking the symmetries of conjunctive normal form (CNF) formulae has been shown to lead to significant reductions in search times. Symmetries in the search space are broken by adding appropriate symmetry-breaking predicates (SBPs) to an SAT instance in CNF. The SBPs prune the search space by acting as a filter that confines the search to nonsymmetric regions of the space without affecting the satisfiability of the CNF formula. For symmetry breaking to be effective in practice, the computational overhead of generating and manipulating SBPs must be significantly less than the runtime savings they yield due to search space pruning. In this paper, we describe a more systematic and efficient construction of SBPs. In particular, we use the cycle structure of symmetry generators, which typically involve very few variables, to drastically reduce the size of SBPs. Furthermore, our new SBP construction grows linearly with the number of relevant variables as opposed to the previous quadratic constructions. Our empirical data suggest that these improvements reduce search runtimes by one to two orders of magnitude on a wide variety of benchmarks with symmetries.


Index Terms-Backtrack Search, clause learning, conjunctive normal form (CNF), graph automorphism, satisfiability (SAT), symmetries.

## 1 Introduction

MODERN Boolean satisfiability (SAT) solvers, based on backtrack search, are now capable of attacking instances with thousands of variables and millions of clauses [22] and are being routinely deployed in a wide range of industrial applications [2], [4], [11], [14], [20]. Their success can be credited to a combination of recent algorithmic advances and carefully tuned implementations [3], [8], [10], [13], [18], [23]. Still, there are problem instances that remain beyond the reach of most SAT solvers.

One aspect of intractability is the presence of symmetry in the conjunctive normal form (CNF) of an SAT instance. Intuitively, the symmetry of a discrete object is a transformation, e.g., a permutation, of its components that leaves the object intact. The symmetries of a CNF formula are permutations of its literals (variables and their negations) that result in a reordering of its clauses (and the literals within clauses) without changing the formula itself. Such symmetries induce an equivalence relation on the set of variable assignments such that two assignments are equivalent if and only if the formula assumes the same truth value (either 0 or 1 ) at each of these assignments. A search algorithm that is oblivious to the existence of these symmetries may end up, wastefully, exploring a set of

- F.A. Aloul is with the Computer Engineering Department. American University of Sharjah, PO Box 26666, Sharjah, UAE.
E-mail: faloul@aus.edu.
- K.A. Sakallah and I.L. Markov are with the Electrical Engineering and Computer Science Department, University of Michigan, 1301 Beal Avenue, Ann Arbor, MI 48109-2122.
E-mail: \{karem, imarkov\}@umich.edu.
Manuscript received 5 Jan. 2005; revised 31 Aug. 2005; accepted 15 Nov. 2005; published online 22 Mar. 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0001-0105.
equivalent unsatisfying assignments before moving on to a more promising region of the search space. On the other hand, knowledge of the symmetries can be used to significantly prune the search space. Symmetries are studied in abstract algebra in terms of groups. We assume the reader to be familiar with the basics of group theory; in particular, we assume familiarity with permutation groups and their representation in terms of irredundant sets of generators. A good reference on the subject is [9].

The rest of the paper is organized into five sections. Section 2 provides a brief review of permutations and permutation groups. Section 3 describes pervious work on symmetry breaking for SAT. Our main contribution on efficient constructions of symmetry-breaking predicates is detailed in Section 4. These constructions are evaluated empirically in Section 5 and we end with conclusions in Section 6.

## 2 Notation and Preliminaries

We will be concerned with permutations on the literals of a set of $n$ Boolean variables, $\left\{x_{1}, \ldots, x_{n}\right\}$, which we assume to be totally ordered according to $x_{1}<x_{2}<\ldots<x_{n}$. We use $I_{n}$ to denote the set of integers between 1 and $n$ inclusive and denote nonempty subsets of $I_{n}$ by uppercase "index variables" $I$ and $J$ as appropriate. Given an index set $I$ and an index $i \in I$, we define the "index selector" functions:

$$
\begin{gather*}
\operatorname{pred}(i, I)=\{j \in I \mid j<i\},  \tag{1}\\
\operatorname{prev}(i, I)=\max (\operatorname{pred}(i, I)),  \tag{2}\\
\operatorname{succ}(i, I)=\{j \in I \mid j>i\}, \tag{3}
\end{gather*}
$$

$$
\begin{equation*}
\operatorname{next}(i, I)=\min (\operatorname{succ}(i, I)) \tag{4}
\end{equation*}
$$

where $\min$ and max return, respectively, the least and greatest element in the given index set. For completeness, we also let $\min (\emptyset)=n+1$ and $\max (\emptyset)=0$.

A permutation $\pi$ of the set of $2 n$ literals $L=$ $\left\{x_{1}, x_{1}^{\prime}, \ldots, x_{n}, x_{n}^{\prime}\right\}$ (where $x_{i}^{\prime}$ denotes the logical negation of $x_{i}$ ) is a function $\pi: L \rightarrow L$ that is both one-to-one and onto. We will denote that $x_{j}$ is the image of $x_{i}$ under $\pi$ by writing $x_{j}=x_{i}^{\pi}$. To preserve Boolean consistency, whenever $\pi$ maps $x_{i}$ to $x_{j}$, it must simultaneously map $x_{i}^{\prime}$ to $x_{j}^{\prime}$. Such implied mappings will be assumed whenever not explicitly specified. A permutation $\pi$ is a phase-shift permutation if $x_{i}^{\pi}=x_{i}^{\prime}$ for some $i \in I_{n}$, i.e., $\pi$ maps some literal to its complement.

Permutations will be expressed either in tabular form or in cyclic notation. For example,

$$
\begin{equation*}
\pi=\binom{x_{1} x_{2} \ldots x_{n}}{x_{1}^{\pi} x_{2}^{\pi} \ldots x_{n}^{\pi}} \tag{5}
\end{equation*}
$$

denotes a permutation that maps $x_{1}$ to $x_{1}^{\pi}$, etc. The same permutation can be expressed as a set of disjoint cycles, such as,

$$
\begin{equation*}
\pi=\left(x_{i}, x_{i}^{\pi},\left(x_{i}^{\pi}\right)^{\pi}, \ldots\right)\left(x_{j}, x_{j}^{\pi},\left(x_{j}^{\pi}\right)^{\pi}, \ldots\right) \ldots \tag{6}
\end{equation*}
$$

Here, a cycle $(a, b, \ldots, z)$ is a shortcut for " $a$ maps to $b, b$ maps to $c, \ldots$, and $z$ maps to $a .{ }^{\prime \prime}$ The length of a cycle is equal to the number of literals in it; we will refer to a cycle whose length is $k$ as a $k$-cycle. We define the support of a permutation $\pi, \operatorname{supp}(\pi)$, to be the set of indices appearing in its cyclic representation, i.e.,

$$
\begin{equation*}
\operatorname{supp}(\pi)=\left\{i \in I_{n} \mid x_{i}^{\pi} \neq x_{i}\right\} \tag{7}
\end{equation*}
$$

The number of cycles in a permutation $\pi$ will be denoted by $\operatorname{cycles}(\pi)$. We also define phase-shift $(\pi)$ to be the index of the smallest variable (according to the assumed total ordering) that is mapped to its complement by $\pi$ :

$$
\begin{equation*}
\text { phase-shift }(\pi)=\min \left\{i \in I_{n} \mid x_{i}^{\pi}=x_{i}^{\prime}\right\} \tag{8}
\end{equation*}
$$

We should note that a phase-shift permutation must have one or more phase-shift cycles, i.e., length-2 cycles that have the form $\left(x_{i}, x_{i}^{\prime}\right)$. Finally, we define $\operatorname{ends}(\pi)$ as follows:

$$
\operatorname{ends}(\pi)=\left\{\begin{array}{ll}
i \in I_{n} \mid i & \text { is the largest index of a variable }  \tag{9}\\
\text { in a non-phase-shift cycle of } \pi
\end{array}\right\} .
$$

A permutation group $G$ is a group whose elements are permutations of some finite set and whose binary operation is function composition, also referred to as permutation multiplication. The order of a group is the number of its elements. A subgroup $H$ of a group $G$, denoted $H \leq G$, is a subset of $G$ that is closed under the group's binary operation. The cyclic subgroup of $\pi \in G$, denoted by $\langle\pi\rangle$, is the subgroup consisting of $\pi$ and its integer powers:

$$
\begin{equation*}
<\pi>=\left\{\pi^{i} \mid(i \in Z)\right\} \tag{10}
\end{equation*}
$$

and $\pi$ is said to generate $<\pi>$. A set of permutations $\pi_{1} \in$ $G, \ldots, \pi_{k} \in G$ generates $G$ if the subgroup resulting from taking all possible products of the integer powers of these permutations is equal to $G$. The permutations $\left\{x_{1}, \ldots, x_{k}\right\}$ are called generators of $G$. A set of generators is irredundant if


Fig. 1. Conversion of the CNF formula $\left(x_{1}^{\prime}+x_{2}+x_{3}\right)\left(x_{1}+x_{2}^{\prime}+x_{3}^{\prime}\right)\left(x_{2}^{\prime}+\right.$ $x_{3}$ ) to a graph for symmetry extraction. Different shapes correspond to different vertex colors.
it is not possible to express any of its permutations as a product of powers of its other permutations. A set of irredundant generators serves as an implicit representation of the group it generates and, in general, guarantees exponential compression in the size of the representation. Note that a set of irredundant generators is not a group since it is not closed under multiplication and taking inverse. In the sequel, a set of permutations $G$ that is not necessarily closed will be indicated by placing a "hat" on the variable denoting the set, i.e., $\hat{G}$. Additionally, and with a slight abuse of notation, we will indicate that $G$ is the group generated by $\hat{G}$ by writing $G=<\hat{G}>$.

## 3 Previous Work

The basic framework for utilizing the symmetries in a CNF instance to prune the search space explored by an SAT solver was laid out in [5]. This framework was extended later, in [1], to account for phase-shift symmetries, take advantage of the cycle structure of permutations, and consider only generators of the group of symmetries. In outline, the procedure consists of the following steps:

1. Convert a CNF formula $\varphi$ to a colored graph whose symmetries are isomorphic to the symmetries of the formula. A simple construction represents every nonbinary clause by a vertex of color 2 and every variable by two vertices of color 1 (one for the positive and one for the negative literal) connected by Boolean consistency edges. Every literal in the CNF formula is then represented by a bipartite edge. Binary clauses are represented by connecting their literal vertices directly [1], [5]. An example is shown in Fig. 1.
2. Find the symmetries of the graph in terms of a set of irredundant generators $\hat{G}=\left\{\pi_{1}, \ldots, \pi_{k}\right\}$ using a suitable graph automorphism program [6], [12], [19].
3. Map the graph symmetries back to symmetries of the CNF formula.
4. Construct an appropriate symmetry-breaking predicate (SBP) $\rho$ and conjoin it to the formula.
5. Solve $\varphi \wedge \rho$ using a suitable SAT solver [13].

Our concern in this paper is Step 4. Noting that the group of symmetries induces an equivalence relation on the set of assignments in the $n$-dimensional Boolean space, the basic idea is to construct a "filter" that picks out a single representative from each equivalence class. In particular, choosing the lexicographically smallest representative-according to the assumed total ordering on the variables-leads to the following Lex-Leader SBP [5]:

$$
\begin{gathered}
\pi=\binom{x_{1} x_{2} x_{3} x_{4} x_{5} x_{6} x_{7} x_{8} x_{9} x_{10}}{x_{4} x_{2} x_{8} x_{1} x_{5}{ }^{\prime} x_{3}{ }^{\prime} x_{7} x_{6}{ }^{\prime} x_{9} x_{10}} \\
\pi=\left(x_{1}, x_{4}\right)\left(x_{3}, x_{8}, x_{6}^{\prime}\right)\left(x_{5}, x_{5}^{\prime}\right)
\end{gathered}
$$

(a)
$\operatorname{supp}(\pi)=\{1,3,4,5,6,8\}$
phase-shift $(\pi)=5$
$\operatorname{succ}\left(\right.$ phase-shift $\left.(\pi), I_{10}\right)=\{6,7,8,9,10\}$
ends $(\pi)=\{4,8\}$
$\operatorname{supp}(\pi) \backslash \operatorname{ends}(\pi)=\{1,3,5,6\}$
$\operatorname{supp}(\pi) \backslash$ ends $(\pi) \backslash \operatorname{succ}\left(\operatorname{phase}-\operatorname{shift}(\pi), I_{10}\right)=\{1,3,5\}$
(b)

$$
\begin{aligned}
& \operatorname{BP}(\pi, 1)=x_{1} \leq x_{4} \\
& \operatorname{BP}(\pi, 2)=\left(x_{1}=x_{4}\right) \rightarrow\left(x_{2} \leq x_{2}\right) \\
& \operatorname{BP}(\pi, 3)=\left(x_{1}=x_{4}\right)\left(x_{2}=x_{2}\right) \rightarrow\left(x_{3} \leq x_{8}\right) \\
& \operatorname{BP}(\pi, 4)=\left(x_{1}=x_{4}\right)\left(x_{2}=x_{2}\right)\left(x_{3}=x_{8}\right) \rightarrow\left(x_{4} \leq x_{1}\right) \\
& \operatorname{BP}(\pi, 5)=\left(x_{1}=x_{4}\right)\left(x_{2}=x_{2}\right)\left(x_{3}=x_{8}\right)\left(x_{4}=x_{1}\right) \rightarrow\left(x_{5} \leq x_{5}^{\prime}\right) \\
& \operatorname{BP}(\pi, 6)=\left(x_{1}=x_{4}\right)\left(x_{2}=x_{2}\right)\left(x_{3}=x_{8}\right)\left(x_{4}=x_{1}\right)\left(x_{5}=x_{5}^{\prime}\right) \rightarrow\left(x_{6} \leq x_{3}^{\prime}\right)
\end{aligned}
$$

$$
\operatorname{BP}(\pi, 7)=\left(x_{1}=x_{4}\right)\left(x_{2}=x_{2}\right)\left(x_{3}=x_{8}\right)\left(x_{4}=x_{1}\right)\left(x_{5}=x_{5}^{\prime}\right)\left(x_{6}=x_{3}^{\prime}\right) \rightarrow\left(x_{7} \leq x_{7}\right)
$$

$$
\operatorname{BP}(\pi, 8)=\left(x_{1}=x_{4}\right)\left(x_{2}=x_{2}\right)\left(x_{3}=x_{8}\right)\left(x_{4}=x_{1}\right)\left(x_{5}=x_{5}^{\prime}\right)\left(x_{6}=x_{3}^{\prime}\right)\left(x_{7}=x_{7}\right) \rightarrow\left(x_{8} \leq x_{6}^{\prime}\right)
$$

$$
\operatorname{BP}(\pi, 9)=\left(x_{1}=x_{4}\right)\left(x_{2}=x_{2}\right)\left(x_{3}=x_{8}\right)\left(x_{4}=x_{1}\right)\left(x_{5}=x_{5}^{\prime}\right)\left(x_{6}=x_{3}^{\prime}\right)\left(x_{7}=x_{7}\right)\left(x_{8}=x_{6}^{\prime}\right) \rightarrow\left(x_{9} \leq x_{9}\right)
$$

$$
\operatorname{BP}(\pi, 10)=\left(x_{1}=x_{4}\right)\left(x_{2}=x_{2}\right)\left(x_{3}=x_{8}\right)\left(x_{4}=x_{1}\right)\left(x_{5}=x_{5}^{\prime}\right)\left(x_{6}=x_{3}^{\prime}\right)\left(x_{7}=x_{7}\right)\left(x_{8}=x_{6}^{\prime}\right)\left(x_{9}=x_{9}\right) \rightarrow
$$

$$
\left(x_{10} \leq x_{10}\right)
$$

(c)

$$
\begin{aligned}
\operatorname{PP}(\pi) & =\left(p_{1}\right)\left(p_{1} \rightarrow l_{1} p_{3}\right)\left(p_{3} \rightarrow g_{1} \rightarrow l_{3} p_{5}\right)\left(p_{5} \rightarrow g_{3} \rightarrow l_{5}\right) \\
& =\left(p_{1}\right)\left(p_{1} \rightarrow\left(x_{1} \leq x_{4}\right) p_{3}\right)\left(p_{3} \rightarrow\left(x_{1} \geq x_{4}\right) \rightarrow\left(x_{3} \leq x_{8}\right) p_{5}\right)\left(p_{5} \rightarrow\left(x_{3} \geq x_{8}\right) \rightarrow x_{5}^{\prime}\right)
\end{aligned}
$$

(d)

Fig. 2. Illustration of different formulations of the permutation predicate. (a) Permutation in tabular and cyclic notation. (b) Various index sets associated with permuation. (c) Bit predicates according to (10). BPs enclosed in boxes with square corners are tautologous because $\pi$ maps the corresponding bits to themselves. BPs enclosed in boxes with rounded corners are tautologous because they correspond to cycle "ends." The BPs for bits 6 to 10 are tautologous because $\pi$ maps bit 5 to its complement. (d) Linear formulation of the permutation predicate according to (18), based only on irredundant bits.

$$
\begin{gather*}
\rho^{L L}(<\hat{G}>)=\bigcap_{\pi \in<\hat{G}>} P P(\pi),  \tag{11}\\
P P(\pi)=\bigcap_{i \in I} B P(\pi, i),  \tag{12}\\
B P(\pi, i)=\left[\bigcap_{j \in \operatorname{pred}(i, I)}\left(x_{j}=x_{j}^{\pi}\right)\right] \rightarrow\left(x_{i} \leq x_{i}^{\pi}\right), \tag{13}
\end{gather*}
$$

where the index set $I$ in (12) and (13) is equal to $I_{n}$. In these equations, the Lex-Leader SBP is expressed as a conjunction of permutation predicates (PPs), each of which is a conjunction of bit predicates (BPs). ${ }^{1}$ Introducing $n$ auxiliary "equality" variables, $e_{i} \equiv\left(x_{i}=x_{i}^{\pi}\right)$, makes it possible to express the $i$ th BP as an $(i+1)$-literal CNF clause. This leads to a

[^0]CNF representation of the PP in (12) that has $n$ clauses with a total literal count of $0.5\left(n^{2}+3 n\right)$. Additionally, each of the introduced equality constraints yields four 3-literal clauses bringing the total CNF size of (12) to:

$$
\begin{align*}
& \operatorname{clauses}(P P(\pi))=5 n \\
& \operatorname{literals}(P P(\pi))=0.5\left(n^{2}+27 n\right) \tag{14}
\end{align*}
$$

In its present form, the lex-leader SBP in (11)-(13) can lead to an exponentially large CNF formula because the order of the symmetry group can be exponential in the number of variables. Thus, its value in pruning the search space is negated by the need of the SAT solver to process a much larger CNF formula. To remedy this problem, the authors of [5] suggested the construction of a symmetry tree to eliminate some redundant permutations. However, in the worst case, the number of symmetries in the tree remains

TABLE 1
Symmetry Statistics for Various Benchmark Families


PS: \# phase-shift generators; Isupp/ Sum: sum of support of generators; S: sparsity of generators (1-Isuppl/(Vars*Gen)).
exponential. Empirical evidence in [1] showed that full symmetry breaking, i.e., insuring that the SBP selects only the lex-leader from each equivalence class, is not necessary to obtain significant pruning of the search space. An SBP that breaks some, but not necessarily all, of the symmetries of the formula can, in fact, provide a much better space/time tradeoff during the search. This is accomplished by replacing the group of symmetries in (11) by a suitable, and much smaller, set of permutations $\hat{H} \subseteq<\hat{G}>$ :

$$
\begin{equation*}
\rho^{L L}(\hat{H})=\bigcap_{\pi \in \hat{H}} P P(\pi) . \tag{15}
\end{equation*}
$$

In particular, the approach in [1] advocated the use of the set of generators $G$ returned by the graph automorphism program in Step 2.

## 4 Efficient Formulation of Permutation Predicate

Even when only a small number of permutations is used in constructing an SBP, as in (15), the corresponding CNF formula may still be too large because each PP requires a CNF formula whose size is quadratic in the number of variables $n$. In this section, we introduce two refinements that lead to much smaller PPs. The first refinement utilizes the cycle structure of a permutation to eliminate redundant bit predicates and can be viewed as replacing $n$ in (14) by a much smaller number $m$ and represents a more comprehensive and systematic treatment of cycles than that in [1]. The second refinement takes advantage of the recursive bit-by-bit structure in (13) to yield a CNF formula whose size is

TABLE 2
Size Comparisons of Three SBP Constructions Based on Group Generators

| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Benchmark Family | Instance | Total SBP Sizes |  |  |  |  |  |  |  |  |
|  |  | All Bits |  |  | Irredundant Bits |  |  |  |  |  |
|  |  | Quadratic Construction |  |  |  |  |  | Linear Construction |  |  |
|  |  | \#Extra Var | \#Clauses | \#Literals | \#Extra Var | \#Clauses | \#Literals | \#Extra Var | \#Clauses | \#Literals |
| $\begin{aligned} & \text { 5 } \\ & \frac{0}{0} \\ & \hline 10 \end{aligned}$ | hole07 | 728 | 3640 | 30212 | 97 | 485 | 1673 | 97 | 362 | 1215 |
|  | hole08 | 1080 | 5400 | 53460 | 127 | 635 | 2254 | 127 | 478 | 1613 |
|  | hole09 | 1530 | 7650 | 89505 | 161 | 805 | 2938 | 161 | 610 | 2067 |
|  | hole10 | 2090 | 10450 | 143165 | 199 | 995 | 3731 | 199 | 758 | 2577 |
|  | hole11 | 2772 | 13860 | 220374 | 241 | 1205 | 4639 | 241 | 922 | 3143 |
|  | hole12 | 3588 | 17940 | 328302 | 287 | 1435 | 5668 | 287 | 1102 | 3765 |
| 5 | Urq3_1 | 1118 | 5590 | 39130 | 0 | 26 | 26 | 0 | 26 | 26 |
|  | Urq3_4 | 684 | 3420 | 21546 | 0 | 19 | 19 | 0 | 19 | 9 |
|  | Urq3_5 | 1334 | 6670 | 48691 | 0 | 29 | 29 | 0 | 29 | 29 |
|  | Urq3_9 | 740 | 3700 | 23680 | 0 | 20 | 20 | 0 | 20 | 20 |
|  | grout3.3-03 | 27840 | 139200 | 13739040 | 720 | 3600 | 20280 | 720 | 2822 | 9761 |
|  | grout3.3-04 | 24624 | 123120 | 11560968 | 600 | 3000 | 15888 | 600 | 2346 | 8103 |
|  | grout3.3-10 | 29568 | 147840 | 16011072 | 720 | 3600 | 20136 | 720 | 2824 | 9772 |
|  | fpga10.08 | 2640 | 13200 | 194040 | 256 | 1280 | 6257 | 256 | 980 | 3342 |
|  | fpga10.09 | 3105 | 15525 | 251505 | 223 | 1115 | 4228 | 223 | 846 | 2869 |
|  | fpga12.08 | 3456 | 17280 | 295488 | 312 | 1560 | 8136 | 312 | 1200 | 4104 |
|  | fpga12.09 | 4050 | 20250 | 382725 | 273 | 1365 | 5300 | 273 | 1042 | 3547 |
|  | fpga12.11 | 5742 | 28710 | 645975 | 339 | 1695 | 6821 | 339 | 1298 | 4427 |
|  | fpga12.12 | 6912 | 34560 | 839808 | 480 | 2400 | 14904 | 480 | 1856 | 6368 |
|  | fpga 13.09 | 4576 | 22880 | 464464 | 299 | 1495 | 5875 | 299 | 1144 | 3900 |
|  | fpga13.10 | 5460 | 27300 | 606060 | 334 | 1670 | 6677 | 334 | 1280 | 4368 |
|  | fpga 13.12 | 7488 | 37440 | 977184 | 406 | 2030 | 8405 | 406 | 1560 | 5332 |
| $\begin{aligned} & \text { 巳 } \\ & \text { O} \\ & \text { O} \\ & \text { U } \end{aligned}$ | chnl10.11 | 8580 | 42900 | 1059630 | 508 | 2540 | 14997 | 508 | 1954 | 6683 |
|  | chnl10.12 | 9840 | 49200 | 1313640 | 556 | 2780 | 17102 | 556 | 2142 | 7333 |
|  | chnl10.13 | 11180 | 55900 | 1604330 | 604 | 3020 | 19325 | 604 | 2330 | 7983 |
|  | chnl11.12 | 11352 | 56760 | 1651716 | 614 | 3070 | 19772 | 614 | 2370 | 8123 |
|  | chnl11.13 | 12870 | 64350 | 2014155 | 667 | 3335 | 22371 | 667 | 2578 | 8843 |
|  | chnl11.20 | 25960 | 129800 | 6061660 | 1038 | 5190 | 44512 | 1038 | 4034 | 13883 |
| oü | x1_16 | 736 | 3680 | 26864 | 1 | 20 | 29 | 1 | 17 | 18 |
|  | x1_24 | 1680 | 8400 | 81480 | 0 | 24 | 24 | 0 | 24 | 24 |
|  | x1_32 | 3008 | 15040 | 181984 | 0 | 32 | 32 | 0 | 32 | 32 |
| $\stackrel{0}{2}$ | 2pipe_1.000 | 2502 | 12510 | 1077111 | 319 | 1597 | 54556 | 319 | 1274 | 4445 |
|  | 2pipe_2.000 | 4625 | 23125 | 2201500 | 358 | 1794 | 67501 | 358 | 1430 | 4981 |
|  | 2pipe | 6027 | 30135 | 2675988 | 354 | 1774 | 64651 | 354 | 1410 | 4903 |
|  | Total | 239K | 1M | 67M | 11093 | 55640 | 468776 | 11093 | 43119 | 147618 |

Extra Vars: \# of additional variables used in SBPs.
linear, rather than quadratic, in $m$. Fig. 2 provides an example illustrating these refinements.

### 4.1 Elimination of Redundant BPs

Careful analysis of (13) reveals three cases in which a BP is tautologous and, hence, redundant. The first corresponds to bits that are mapped to themselves by the permutation, i.e., $x_{i}^{\pi}=x_{i}$. This makes the consequent of the implication in (13), and, hence, the whole bit predicate, unconditionally true. Removal of such BPs is easily accomplished by setting the index set $I$ in (12) and (13) to $\operatorname{supp}(\pi)$ rather than $I_{n}$. For sparse permutations, i.e., permutations for which $|\operatorname{supp}(\pi)| \ll n$, this change
alone can account for most of the reduction in the CNF size of the PP.

The second case corresponds to the BP of the last bit in each cycle of $\pi$. "Last" here refers to the assumed total ordering on the variables. Assume a cycle involving the variables $\left\{x_{j} \mid j \in J\right\}$ for some index set $J$ and let $i=\max (J)$. Then,

$$
\begin{equation*}
\left[\bigcap_{j \in J \backslash\{i\}}\left(x_{j}=x_{j}^{\pi}\right)\right] \rightarrow\left(x_{i} \leq x_{i}^{\pi}\right)=1 \tag{16}
\end{equation*}
$$

causing the corresponding bit predicate $B P(\pi, i)$ to be tautologous. Elimination of these BPs is accomplished by restricting the index set $I$ in (12) and (13) further to just

TABLE 3
Comparison of Search Runtimes for Various Choices of SBP Constructions and Symmetries to Break

| 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Benchmark Family | Instance |  |  |  |  | Time to solve instances and SBPs (sec) |  |  |  | Speedup |  |
|  |  |  |  |  |  | Generators only |  |  | Generators \& their compositions | $\begin{array}{r} 8 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$ | $\begin{aligned} & \text { ep } \\ & \begin{array}{c} \overline{0} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \\ & \hline 0 \end{aligned}$ |
|  |  |  |  |  |  | All Bits | Irredundant Bits |  |  |  |  |
|  |  |  |  |  |  | Quadratic construction |  | Linear construction |  |  |  |
| $\begin{aligned} & \text { 5 } \\ & \frac{0}{0} \\ & \hline 10 \end{aligned}$ | hole07 | 13 | 102 | 0.00 | 0.03 | 0.03 | 0.01 | 0.01 | 0.01 | 3 | 3 |
|  | hole08 | 15 | 133 | 0.00 | 0.15 | 0.17 | 0.01 | 0.01 | 0.01 | 17 | 15 |
|  | hole09 | 17 | 168 | 0.01 | 0.97 | 0.30 | 0.01 | 0.01 | 0.01 | 15 | 49 |
|  | hole10 | 19 | 207 | 0.02 | 14.4 | 2.87 | 0.01 | 0.01 | 0.01 | 96 | 480 |
|  | hole11 | 21 | 250 | 0.02 | 133 | 9.04 | 0.01 | 0.01 | 0.02 | 301 | 4440 |
|  | hole12 | 23 | 297 | 0.02 | >1000 | 6.90 | 0.01 | 0.01 | 0.03 | 230 | >33333 |
| ¢ | Urq3_1 | 26 | 26 | 0.02 | 159 | 0.20 | 0.04 | 0.04 | 0.01 | 3 | 2650 |
|  | Urq3_4 | 19 | 19 | 0.01 | 0.13 | 0.02 | 0.01 | 0.01 | 0.01 | 1 | 7 |
|  | Urq3_5 | 29 | 29 | 0.04 | >1000 | 0.62 | 0.89 | 0.88 | 0.01 | 1 | >1087 |
|  | Urq3_9 | 20 | 20 | 0.01 | 5.14 | 0.01 | 0.01 | 0.01 | 0.01 | 1 | 257 |
| $\begin{aligned} & \text { پ. } \\ & \text { 訁̈ㅇ } \end{aligned}$ | grout3.3-03 | 29 | 447 | 0.51 | 40.5 | 408 | 1.15 | 0.37 | 3.08 | 464 | 46 |
|  | grout3.3-04 | 27 | 391 | 0.46 | 1.12 | 199 | 0.32 | 0.03 | 1.89 | 262 | 1 |
|  | grout3.3-10 | 28 | 418 | 0.55 | 808 | 331 | 4.38 | 0.52 | 0.67 | 309 | 755 |
|  | fpga 10.08 | 22 | 269 | 0.02 | 24.6 | 0.38 | 0.02 | 0.01 | 0.04 | 13 | 820 |
|  | fpga 10.09 | 23 | 294 | 0.03 | 150 | 1.43 | 0.01 | 0.01 | 0.02 | 36 | 3750 |
|  | fpga12.08 | 24 | 318 | 0.03 | 389 | 1.36 | 0.01 | 0.01 | 0.10 | 34 | 9725 |
|  | fpga 12.09 | 25 | 345 | 0.04 | >1000 | 2.39 | 0.01 | 0.01 | 0.04 | 48 | $>20000$ |
|  | fpga 12.11 | 29 | 459 | 0.07 | >1000 | 4.98 | 0.02 | 0.01 | 0.10 | 62 | >12500 |
|  | fpga 12.12 | 32 | 554 | 0.08 | 989 | 3.27 | 0.01 | 0.01 | 0.22 | 36 | 10989 |
|  | fpga13.09 | 26 | 372 | 0.05 | $>1000$ | 3.75 | 0.02 | 0.02 | 0.05 | 54 | $>14286$ |
|  | fpga 13.10 | 28 | 429 | 0.06 | $>1000$ | 8.48 | 0.02 | 0.01 | 0.10 | 121 | >14286 |
|  | fpga13.12 | 32 | 555 | 0.10 | >1000 | 20.8 | 0.02 | 0.01 | 0.14 | 189 | >9091 |
|  | chnl10.11 | 39 | 814 | 0.06 | 14.5 | 27.8 | 0.04 | 0.01 | 0.07 | 396 | 207 |
|  | chn110.12 | 41 | 897 | 0.08 | 14.7 | 0.75 | 0.05 | 0.01 | 0.08 | 8 | 163 |
|  | chnl10.13 | 43 | 984 | 0.09 | 15.4 | 44.1 | 0.05 | 0.01 | 0.09 | 442 | 154 |
|  | chnl11.12 | 43 | 984 | 0.09 | 133 | 2.34 | 0.07 | 0.01 | 0.11 | 23 | 1334 |
|  | chnl11.13 | 45 | 1075 | 0.11 | 243 | >1000 | 0.08 | 0.01 | 0.12 | 8333 | 2025 |
|  | chnl11.20 | 59 | 1824 | 0.30 | >1000 | 2.78 | 0.16 | 0.01 | 0.30 | 9 | >3226 |
| N | x1_16 | 16 | 16 | 0.01 | 0.03 | 0.01 | 0.01 | 0.01 | 0.01 | 1 | 2 |
|  | x1_24 | 24 | 24 | 0.01 | 12.4 | 0.31 | 0.03 | 0.03 | 0.01 | 8 | 310 |
|  | x1_32 | 32 | 32 | 0.03 |  | 0.05 | 0.01 | 0.01 | 0.01 | 1 | >25000 |
| $\stackrel{0}{2}$ | 2pipe_1.000 | 3 | 6 | 0.28 | 0.21 | 0.99 | 0.23 | 0.27 | 0.28 | 2 | 0 |
|  | 2pipe_2.000 | 5 | 15 | 0.36 | 0.28 | 1.78 | 0.69 | 0.35 | 0.35 | 3 |  |
|  | 2pipe | 7 | 28 | 0.20 | 0.17 | 2.19 | 0.45 | 0.23 | 0.34 | 5 | 0 |
|  | Total | 885 | 14802 | 3.77 | 12149 | 2088 | 8.05 | 3.22 | 8.35 | 339 | 5029 |
|  | Geometric <br> Mean |  |  |  |  |  |  |  |  | 24 | 344 |

Search runtimes are significantly smaller after augmenting the instances with symmetry-breaking predicates.
$\operatorname{supp}(\pi) \backslash e n d s(\pi)$ and corresponds to a reduction in the number of BPs from $n$ to $m \equiv|\operatorname{supp}(\pi)|-\operatorname{cycles}(\pi)$.

The third and last case corresponds to the BPs of those bits that occur after the first "phase-shifted variable." Let $i$ be the index of the first variable for which $x_{i}^{\pi}=x_{i}^{\prime}$. Thus, $e_{i}=0$ and all BPs for $j>i$ have the form $0 \rightarrow\left(x_{j} \leq x_{j}^{\pi}\right)$, making them unconditionally true.

Taken together, the redundant BPs corresponding to these three cases can be easily eliminated by setting the index set in (12) and (13) to:

$$
\begin{equation*}
I=(\operatorname{supp}(\pi) \backslash \operatorname{ends}(\pi)) \backslash \operatorname{succ}\left(\text { phase }-\operatorname{shift}(\pi), I_{n}\right) . \tag{17}
\end{equation*}
$$

In the sequel, we will refer to the bits in the above index set as "irredundant bits." Note that the presence of a phaseshifted variable early in the total order can lead to a drastic reduction in the number of irredundant bits. For example, if $\pi=\left(x_{1}, x_{1}^{\prime}\right) \ldots$, then $P P(\pi)$ is simply $\left(x_{1}^{\prime}\right)$, regardless of how many other variables are moved by $\pi$.

TABLE 4
zChaff Search Runtimes of "Randomized" Hole-n Instances Augmented with Linear SBPs Based on Different Sets of Permutations

|  | 2-cycle generators |  | Generators with long cycles |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | Total | Time (sec) | Max cycle length | Generators only |  | Generators and their powers |  | Generators and their compositions |  |
|  |  |  |  | Total | Time (sec) | Total | Time (sec) | Total | Time (sec) |
| Hole-7 | 13 | 0.01 | 24 | 13 | 0.01 | 35 | 0.02 | 116 | 0.03 |
| Hole-8 | 15 | 0.01 | 21 | 15 | 0.04 | 44 | 0.1 | 149 | 0.13 |
| Hole-9 | 17 | 0.01 | 56 | 17 | 0.41 | 83 | 1.96 | 187 | 1.01 |
| Hole-10 | 19 | 0.01 | 70 | 19 | 2.38 | 243 | 13.68 | 232 | 2.2 |

Total denotes the total number of permutations used in constructing each SBP.

### 4.2 Linear Construction of PPs through Chaining

The PP in (12) and (13) has a recursive structure that can be utilized to produce a CNF formula whose size is linear, rather than quadratic, in the cardinality of the index set I. Specifically, we introduce the "ordering" predicates $l_{i}=$ $\left(x_{i} \leq x_{i}^{\pi}\right)$ and $g_{i}=\left(x_{i} \geq x_{i}^{\pi}\right)$ and, after algebraic manipulation, write the following equivalent expressions for the permutation predicate:

$$
\begin{align*}
P P(\pi) & =g_{0} \rightarrow \bigcap_{i \in I}\left\{\left(\bigcap_{j \in \operatorname{pred}(i, I)} g_{j}\right) \rightarrow l_{i}\right\} \\
& =g_{0} \rightarrow l_{k} \wedge\left[g_{k} \rightarrow \bigcap_{i \in K}\left\{\left(\bigcap_{j \in \operatorname{pred}(i, K)} g_{i}\right) \rightarrow l_{i}\right\}\right], \tag{18}
\end{align*}
$$

where $g_{0}=1, k=\operatorname{next}(0, I)$, and $K=\operatorname{succ}(k, I)$. Noting that, except for the index set used, the parenthesized expression on the second line of the above equation is identical to the expression on the first line, we introduce a sequence of chaining predicates $\left\{p_{i} \mid i \in I\right\}$ defined according to:

$$
\begin{equation*}
p_{i}=g_{\operatorname{prev}(i, I)} \rightarrow \bigcap_{k \in K}\left[\bigcap_{j \in \operatorname{pred}(k, K)} g_{j}\right] \rightarrow l_{k} \tag{19}
\end{equation*}
$$

where $K=\{i\} \cup \operatorname{succ}(i, I)=\{k \in I \mid k \geq i\}$. The recursive structure of (18) now makes it possible to express each chaining predicate in terms of the one that follows it:

$$
\begin{equation*}
p_{i}=g_{\text {prev }(i, I)} \rightarrow l_{i} p_{\text {next }(i, I)} \quad i \in I, p_{n+1} \equiv 1 \tag{20}
\end{equation*}
$$

and yields the following alternative representation of the permutation predicate:

$$
\begin{equation*}
P P(\pi)=p_{\min (I)} \wedge \bigcap_{i \in I}\left[p_{i}=g_{\operatorname{prev}(i, I)} \rightarrow l_{i} p_{\text {next }(i, I)}\right] \tag{21}
\end{equation*}
$$

which can be simplified further by replacing the equalities by one-way implications leading, finally, to:

$$
\begin{equation*}
P P(\pi)=p_{\min (I)} \wedge \bigcap_{i \in I}\left[p_{i} \rightarrow g_{\operatorname{prev}(i, I)} \rightarrow l_{i} p_{\text {next }(i, I)}\right] \tag{22}
\end{equation*}
$$

The CNF representation of each conjunct in (22) is obtained by substituting the definitions of the $l$ and $g$ variables and
using the distributive law. Thus, using this construction, the permutation predicate requires $|I|$ additional variables (the chaining predicates) and consists of $2|I| 3$-literal and $2|I|$ 4 -literal clauses for a total of $14|I|$ literals.

## 5 Experimental Results

We conducted a number of experiments to evaluate the effectiveness of the symmetry breaking constructions described above in reducing search times. We ran the experiments on representative CNF instances from the following six benchmark families:

1. Hole-n: Unsatisfiable pigeon-hole instances [7].
2. Urq: Unsatisfiable randomized instances based on expander graphs [21].
3. GRoute: Difficult satisfiable instances that model global wire routing in integrated circuits [2].
4. FPGARoute and ChnlRoute: Large satisfiable and unsatisfiable instances that model the routing of wires in the channels of field-programmable integrated circuits [14].
5. XOR: Various exclusive-or chains [16].
6. 2pipe: Difficult unsatisfiable instances that model the functional correctness requirements of modern out-of-order microprocessor CPUs [22].
Each of the benchmarks was converted to a colored graph, as described in Section 3, and processed by the graph automorphism program Saucy [6]. The symmetries returned by Saucy were then mapped back to symmetries of the benchmark and appropriate SBPs constructed and added. The zChaff SAT solver [13] was then run on the original and SBP-augmented versions of each benchmark. All experiments were run on a Linux workstation with a 2 Ghz Pentium 4 processor and 1GB of RAM. A time-out limit of 1,000 seconds was set for all runs.

Table 1 lists, for each benchmark family, the name of the tested instance (column 2), its total CNF size (columns 3, 4, and 5), the order of its symmetry group (column 6), the total number of generators returned by Saucy (column 7), and the number of those that include phase shifts (column 8 ). Columns 9 and 10 list the cardinality of the generators' support and the degree of sparsity present in these generators. Table 2 lists the CNF sizes of three SBP constructions based on generators:

TABLE 5
Comparison of Search Runtimes for Various SAT Solvers with and without SBPs


Search runtimes are significantly smaller after augmenting the instances with symmetry-breaking predicates.

- The quadratic construction (using extra equality variables) based on all bits; this represents the previous state-of-the-art.
- The quadratic construction based only on irredundant bits.
- The linear construction (using extra chaining variables) based only on irredundant bits.
Several observations can be made about the data in Tables 1 and 2 . The number of symmetries in these benchmarks is large, but all symmetries, including phase shifts in benchmark families Urq, XOR, and 2pipe, can be
represented by fairly small sets of generators. The generators returned by Saucy appear very sparse on average, i.e., a typical generator affects only a small number of variables. This explains the reduction, by 1-2 orders-ofmagnitude, in the size of symmetry-breaking predicates in column 18 (our first construction) versus column 15 ([5]): The number of variables, clauses, and literals is reduced. While our construction in column 18 only slightly extends the quadratic-size construction in [1], our more advanced linear-size construction (column 21) offers an additional reduction by up to an order of magnitude. Note, however,
that the number of variables is unchanged-the extra variables added by the two constructions have different function, but can be mapped to each other one-to-one.

Table 3 empirically compares the effectiveness of the symmetry-breaking predicates described in Table 1. First, in most cases, it takes much less time to find symmetries of a CNF instance than to solve it. The 2pipe instances are an exception, but we believe that further advances in symme-try-finding can rectify this exception. Second, the all-bits quadratic-sized construction due to [5] is dramatically slower than our variants, based on the cycle structure. Our linear-sized construction provides a further speed-up. The only exception is the 2pipe_1_ooo instance, where the difference between the irredundant-bits linear and quad-ratic-sized constructions is small.

Table 3 offers additional data to evaluate symmetrybreaking by generators, which may not be complete. We added symmetry-breaking predicates built for pairwise products of generators, but the overall runtimes increased in most cases. While additional SBPs may break more symmetries, their overhead does not justify their use.

Table 4 describes experiments with generators that have long cycles in which we evaluated extensions to symmetrybreaking by generators. Namely, we tried adding powers of all generators and, alternatively, adding pairwise products of generators. Neither extension proved useful, which supports our main symmetry-breaking approach.

In order to study the effect of symmetry breaking when using other state-of-the-art SAT solvers, we solved the instances using two of the best known SAT solvers: BerkMin562 [10] and miniSAT v1.14 [8]. The default settings were used with both solvers. We used the advanced linearsize construction for generating the SBPs. Table 5 shows BerkMin's and miniSAT's runtimes when solving the instances with and without symmetry-breaking predicates. For both solvers, the addition of SBPs leads to significant runtime savings ( 8,243 versus 1.37 seconds for BerkMin and 5,367 versus 2.09 for miniSAT).

We noticed that running local search solvers, e.g., WalkSAT [17], with symmetry-breaking clauses does not improve runtimes. In some cases, it makes runtimes worse, which was also observed by Prestwich in [15].

In terms of complexity, the processing of CNF-SAT instances which results in the addition of SBPs includes symmetry-finding, for which no polynomial-time algorithms are currently known in the general case (but the graph automorphism problem solved as a step is not believed to be NP-complete unless $\mathrm{P}=\mathrm{NP}$ ). However, symmetry-finding is often performed very quickly in practice. Given symmetry-generators, we build one SBP per symmetry-generator. One can show that, for a graph with $N$ vertices, the maximal number of symmetrygenerators returned is $N^{2}\left(\log _{2} N\right)^{2}$, therefore, for a CNF instance with $V$ variables and $C$ clauses, the number of irredundant generators is $O\left((V+C)^{2}\left(\log _{2}(V+C)\right)^{2}\right)$. In practice, symmetries are often represented more compactly and the number of generators is much smaller than $V$. For example, all permutational symmetries of $k$ variables can be captured by just two generators.

To estimate the size of an SBP, we build for a given symmetry generator, we only use its action on vertices and ignore the permutation of clauses that it performs. Since vertices that are mapped onto themselves do not affect the size of an SBP, the size is a function of the vertex-based support of the symmetry-generator. While this support may include all vertices, in practice, it is typically much smaller, e.g., many generators are transpositions $(\mid$ supp $\mid=2)$ or small sets of transpositions. When we build a new SBP, the number of literals in added clauses grows linearly with the size of the vertex-based support of the symmetry generator -this is in contrast to quadratically-growing SBPs in the previous literature. Our algorithm to produce SBPs also has linear asymptotics.

## 6 Conclusions

The main contribution of our work is a better construction of symmetry-breaking predicates for Boolean satisfiability. We empirically demonstrate improvements both in the size of predicates and the runtime of SAT solvers after these predicates are added to the original CNF instances. We also show that 1 ) symmetry-breaking by generators is difficult to improve upon and that 2 ) the efficiency of symmetrybreaking does not improve when larger cycles are found in generators.

Our work articulates that better symmetry finding algorithms would be useful, especially if tailored to CNF formulas and, perhaps, the kinds of symmetry groups commonly found in structured CNF instances.

## Acknowledgments

This work was funded by the DARPA/MARCO Gigascale Silicon Research Center and an Agere Systems/SRC Research fellowship.

## References

[1] F. Aloul, A. Ramani, I.L. Markov, and K. Sakallah, "Solving Difficult Instances of Boolean Satisfiability in the Presence of Symmetries," IEEE Trans. Computer Aided Design, vol. 22, no. 9, pp. 1117-1137, 2003.
[2] F. Aloul, A. Ramani, I.L. Markov, and K. Sakallah, "Generic ILP versus Specialized 0-1 ILP," Proc. Int'l Conf. Computer-Aided Design (ICCAD), pp. 450-457, 2002.
[3] R. Bayardo Jr. and R. Schrag, "Using CSP Look-Back Techniques to Solve Real World SAT Instances," Proc. Nat'l Conf. Artificial Intelligence, pp. 203-208, 1997.
[4] A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu, "Symbolic Model Checking Using SAT Procedures instead of BDDs," Proc. Design Automation Conf. (DAC), pp. 317-320, 1999.
[5] J. Crawford, M. Ginsberg, E. Luks, and A. Roy, "SymmetryBreaking Predicates for Search Problems," Proc. Int'l Conf. Principles of Knowledge Representation and Reasoning, pp. 148-159, 1996.
[6] P. Darga, M.H. Liffiton, K.A. Sakallah, and I.L. Markov, "Exploiting Structure in Symmetry Generation for CNF," Proc. Design Automation Conf. (DAC), pp. 530-534, 2004.
[7] DIMACS Challenge benchmarks, ftp://Dimacs.rutgers.EDU/ pub/challenge/sat/benchmarks/cnf, 1996.
[8] N. Een and N. Sorensson, "An Extensible SAT-Solver," Theory and Applications of Satisfiability Testing, pp. 502-518, 2003.
[9] J.B. Fraleigh, A First Course in Abstract Algebra, sixth ed. Reading, Mass.: Addison Wesley Longman, 2000.
[10] E. Goldberg and Y. Novikov, "BerkMin: A Fast and Robust SATSolver," Proc. Design, Automation, and Test in Europe Conf. (DATE), pp. 142-149, 2002.
[11] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability," IEEE Trans. Computer Aided Design, vol. 11, no. 1, pp. 4-15, 1992.
[12] B. McKay, "Practical Graph Isomorphism," Congressus Numerantium, vol. 30, pp. 45-87, 1981, http://cs.anu.edu.au/bdm/ nauty/.
[13] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," Proc. Design Automation Conf. (DAC), pp. 530-535, 2001.
[14] G. Nam, F. Aloul, K. Sakallah, and R. Rutenbar, "A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints," Proc. Int'l Symp. Physical Design (ISPD), pp. 222-227, 2001.
[15] S. Prestwich, "Supersymmetric Modeling for Local Search," Proc. Workshop Symmetry and CSPs (SymCon), 2002.
[16] SAT 2002 Competition, http://www.satlive.org/SATCompetition /submittedbenchs.html, 2002.
[17] B. Selman, H. Kautz, and B. Cohen, "Noise Strategies for Improving Local Search," Proc. Nat'l Conf. Artificial Intelligence, pp. 337-343, 1994.
[18] J. Silva and K. Sakallah, "GRASP: A New Search Algorithm for Satisfiability," IEEE Trans. Computers, vol. 48, no. 5, pp. 506-521, May 1999.
[19] E. Spitznagel, "Review of Mathematical Software, GAP," Notices Am. Math. Soc., vol. 41, no. 7, pp. 780-782, 1994.
[20] P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, "Combinational Test Generation Using Satisfiability," IEEE Trans. Computer-Aided Design, vol. 15, no. 9, pp. 1167-1175, 1996.
[21] A. Urquhart, "Hard Examples for Resolution," J. ACM, vol. 34, no. 1, pp. 209-219, 1987.
[22] M.N. Velev and R.E. Bryant, "Effective Use of Boolean Satisfiability Procedures in the Formal Verification of Superscalar and VLIW Microprocessors," Proc. Design Automation Conf. (DAC), pp. 226-231, 2001.
[23] H. Zhang, "SATO: An Efficient Propositional Prover," Proc. Int'l Conf. Automated Deduction, pp. 272-275, 1997.


Fadi A. Aloul (S'97-M'03) received the BS degree in electrical engineering (summa cum laude) from Lawrence Technological University, Southfield, Michigan, in 1997 and the MS and PhD degrees in computer science and engineering from the University of Michigan, Ann Arbor, in 1999 and 2003, respectively. He was a postdoctoral research fellow at the University of Michigan during the summer of 2003. He is currently an assistant professor of computer engineering at the American University of Sharjah (AUS), Sharjah, United Arab Emirates. In the summer of 2005, he was a visiting researcher with the Advanced Technology Group at Synopsys, Portland, Oregon. He has published more than 40 papers and has presented invited talks at several industrial sites. He has also developed several tools for Boolean satisfiability, including the pseudo-Boolean SAT solver and optimizer PBS. His research interests are in the areas of computeraided design, verification, combinatorial optimization, and Boolean satisfiability. He has received a number of awards, including the Agere/ SRC research fellowship, GANN fellowship, and the LTU presidential scholarship. He served on the technical program committees at the International Workshop on Logic Synthesis (IWLS), the International Conference on Theory and Applications of Satisfiability Testing (SAT), the SIGDA PhD Forum at the Design Automation Conference, the International Workshop on Soft Constraints (CP-Soft), and the International Symposium on Wireless Systems and Networks. He was the AV chair of the 2003 International Workshop on Logic Synthesis (IWLS). He is a member of the IEEE, the ACM, and Tau Beta Pi.


Karem A. Sakallah (S'76-M'81-SM'92-F'98) received the $B E$ degree in electrical engineering from the American University of Beirut, Beirut, Lebanon, in 1975, and the MSEE and PhD degrees in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, Pennsylvania, in 1977 and 1981, respectively. In 1981, he was with the Department of Electrical Engineering at Carnegie Mellon University as a visiting assistant professor. From 1982 to 1988, he was with the Semiconductor Engineering Computer-Aided Design Group at Digital Equipment Corporation in Hudson, Massachusetts, where he headed the Analysis and Simulation Advanced Development Team. Since September 1988, he has been with the University of Michigan, Ann Arbor, as a professor of electrical engineering and computer science. From September 1994 to March 1995, he was with the Cadence Berkeley Laboratory in Berkeley, California, on a six-month sabbatical leave. He has authored or coauthored more than 200 papers and has presented seminars and tutorials at many professional meetings and various industrial sites. His current research interests include the area of computer-aided design with emphasis on logic and layout synthesis, Boolean satisfiability, discrete optimization, and hardware and software verification. Dr. Sakallah was an associate editor of the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems from 1995 to 1997 and has served on the program committees of the International Conference on Computer-Aided Design, Design Automation Conference, and the International Conference of Computer Design as well as numerous other workshops. He is currently an associate editor of the IEEE Transactions on Computers. He is a fellow of the IEEE and a member of the ACM and Sigma Xi.


Igor L. Markov (M'97-SM'05) received the MA degree in mathematics (1994) and the PhD degree in computer science (2001), both from the University of California Los Angeles. He is currently an assistant professor in the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor. He published more than 100 refereed papers in journals, magazines, and conference proceedings. His interests include combinatorial optimization with applications to the design and verification of integrated circuits, as well as quantum logic circuits. He is a senior member of the IEEE and a member of the ACM and the AMS. In 2001, he was awarded the DAC Fellowship and received the IBM University Partership Award. He received the 2004 IEEE CAS Donald O. Pederson paper-of-the-year award and the 2004 ACM SIGDA Outstanding New Faculty Award. He won the best paper award at DATE 2005 in the Circuit Test category, the US National Science Foundation Career Award, the Synplicity Inc. Faculty Award, and the 2005 SCM SIGDA Technical Leadership Award. He served on the technical program committees at the Design Automation Conference, International Conference on Computer-Aided Design, Design Automation and Test in Europe Conference, International Symposium on Physical Design, and several other IEEE conferences and symposia. He served as the general chair and the technical program committee chair of the International Workshop on System-Level Interconnect Prediction. Currently, he is serving as a guest editor of Integration: The VLSI Journal
$>$ For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.


[^0]:    1. Note that $x \leq y$ in the bit predicate mean "x implies y ."
